A specialized ODE integrator for the efficient computation of parameter sensitivities

Gonnet, Pedro ; Dimopoulos, Sotiris ; Widmer, Lukas ; Stelling, Jörg

Abstract: BACKGROUND: Dynamic mathematical models in the form of systems of ordinary differential equations (ODEs) play an important role in systems biology. For any sufficiently complex model, the speed and accuracy of solving the ODEs by numerical integration is critical. This applies especially to systems identification problems where the parameter sensitivities must be integrated alongside the system variables. Although several very good general purpose ODE solvers exist, few of them compute the parameter sensitivities automatically. RESULTS: We present a novel integration algorithm that is based on second derivatives and contains other unique features such as improved error estimates. These features allow the integrator to take larger time steps than other methods. In practical applications, i.e. systems biology models of different sizes and behaviors, the method competes well with established integrators in solving the system equations, and it outperforms them significantly when local parameter sensitivities are evaluated. For ease-of-use, the solver is embedded in a framework that automatically generates the integrator input from an SBML description of the system of interest. CONCLUSIONS: For future applications, comparatively 'cheap' parameter sensitivities will enable advances in solving large, otherwise computationally expensive parameter estimation and optimization problems. More generally, we argue that substantially better computational performance can be achieved by exploiting characteristics specific to the problem domain; elements of our methods such as the error estimation could find broader use in other, more general numerical algorithms.

DOI: https://doi.org/10.1186/1752-0509-6-46

Gonnet *et al. BMC Systems Biology* 2012, **6**:46
http://www.biomedcentral.com/1752-0509/6/46

BMC Systems Biology

METHODOLOGY ARTICLE Open Access

A specialized ODE integrator for the efficient computation of parameter sensitivities

Pedro Gonnet[1,2,3], Sotiris Dimopoulos[2], Lukas Widmer[2] and Jörg Stelling[2*]

Abstract

Background: Dynamic mathematical models in the form of systems of ordinary differential equations (ODEs) play an important role in systems biology. For any sufficiently complex model, the speed and accuracy of solving the ODEs by numerical integration is critical. This applies especially to systems identification problems where the parameter sensitivities must be integrated alongside the system variables. Although several very good general purpose ODE solvers exist, few of them compute the parameter sensitivities automatically.

Results: We present a novel integration algorithm that is based on second derivatives and contains other unique features such as improved error estimates. These features allow the integrator to take larger time steps than other methods. In practical applications, i.e. systems biology models of different sizes and behaviors, the method competes well with established integrators in solving the system equations, and it outperforms them significantly when local parameter sensitivities are evaluated. For ease-of-use, the solver is embedded in a framework that automatically generates the integrator input from an SBML description of the system of interest.

Conclusions: For future applications, comparatively 'cheap' parameter sensitivities will enable advances in solving large, otherwise computationally expensive parameter estimation and optimization problems. More generally, we argue that substantially better computational performance can be achieved by exploiting characteristics specific to the problem domain; elements of our methods such as the error estimation could find broader use in other, more general numerical algorithms.

Keywords: Dynamical models, ordinary differential equations, parameter sensitivities, integration

Background

In systems biology, mathematical models often take the form of system of ordinary differential equations (ODEs). These are approximations of the underlying mechanisms such as enzyme-catalyzed biochemical reactions that are applicable when molecule numbers are sufficiently high, and when the spatial distributions of components in a cell can be neglected. More specifically, ODE models consider the rate of change in a set of states (*e.g.* species concentrations) as a function of the system's current state, its inputs, and its inherent kinetic parameters that capture, for instance, affinities of molecular interactions [1].

In contrast to systems modeling in domains such as physics, however, model parameters and initial conditions for systems biology models are often not known, or they can only be roughly approximated. As few kinetic parameters can be measured directly, parametric uncertainty often prevails [2]. How the system variables depend on these *system parameters* can therefore be of interest, *e.g.* to help find parameters such that the simulated system matches some observed or desired behavior. Dependencies between system variables and parameters are captured by the *local parameter sensitivities* that describe to what extent the state of the system changes when parameter values are perturbed from a reference value. Formally, local parameter sensitivities comprise the set of derivatives of all system variables with respect to the system parameters. As with the state dynamics, the parameter sensitivities' time evolution follows a system of ODEs [3].

From a computational point of view it is important to note that in all but the simplest cases, starting from a set of initial conditions, there is no direct way to compute the

*Correspondence: joerg.stelling@bsse.ethz.ch
[2]Department of Biosystems Science and Engineering, ETH Zurich, 8092 Zürich, Switzerland
Full list of author information is available at the end of the article

solutions of a system of ODEs (states or parameter sensitivities in our case) for an arbitrary time. The variables are therefore integrated numerically in small steps over time, until the desired end time is reached. Consequently, efficient and accurate numerical integration methods are critical for many applications.

The computational effort for numerical integration is linked to the system size, and over time mathematical models have become increasingly detailed to achieve better predictions. Nevertheless, even models of moderate complexity result in numerical challenges when parameter sensitivities are needed. For instance, the parameter sensitivities can be integrated naively alongside the system variables, but this implies integrating a system of size $n_x \times (1 + n_p)$, where n_x and n_p are the number of system variables and system parameters respectively [3].

Additionally, the solution of a system of ODEs is often used in system identification processes where global optimization or probabilistic inference are required [4,5]. In such cases, thousands, if not millions of trajectories need to be computed. Assessing the quality of the identified model, for instance with respect to the uncertainty in parameter values, again requires computing the local parameter sensitivities [6]. Although local sensitivity information can often help improve the overall estimation process, sensitivity computations are rarely included for performance reasons. Specific efficient methods exist for cases in which only scalar valued functionals are optimized [7] or oscillatory systems are considered [8]. Yet in many other cases, such as optimal control [9,10], the identification of relevant parameters [11], model reduction and simplification [12] or parameter training [13], the full parameter sensitivities need to be computed. Consequently, improvements with respect to the speed with which the original ODE systems and their parameter sensitivities can be reliably integrated may affect the entire process significantly.

These issues are not new and they concern many application domains. Considerable efforts have been invested in establishing reliable and efficient general-purpose ODE solvers for dynamic systems and—to a lesser extent—for the associated parameter sensitivities. Here, however, we are concerned with solving systems of ODEs as they typically occur in the simulation of biochemical reaction networks in systems biology [6]. We show that rather general characteristics of such systems allow for the development of application domain-oriented ODE solvers with novel numerical features (with potential broader applicability), and with superior performance compared to state-of-the-art, widely employed general-purpose solvers. To provide some context for this claim, we first briefly review key characteristics of systems biology models in the form of ODEs, and general methods for the numerical integration of ODEs.

Dynamic models of (bio)chemical networks

When the effects of stochastic noise and of discrete molecule numbers are negligible, ODE systems can be used to describe chemical or biological reaction networks. The n_x time-dependent state variables $x_i(t, \mathbf{p})$, $i = 1 \dots n_x$, which represent the concentrations of the molecules of interest at time t and are usually known at some initial time $t = t_0$, evolve following

$$\dot{\mathbf{x}}(t, \mathbf{p}) := \frac{\partial \mathbf{x}(t, \mathbf{p})}{\partial t} = \mathbf{f}(\mathbf{x}(t), \mathbf{p}), \quad \mathbf{x}(t_0, \mathbf{p}) = \mathbf{x}_0 \tag{1}$$

where $\mathbf{f}(\mathbf{x}(t), \mathbf{p})$ is a system of functions $f_i(\mathbf{x}(t), \mathbf{p})$ modelling the conversion rate of each respective variable $x_i(t, \mathbf{p})$ at time t, and $\mathbf{p}$ is a vector of n_p system parameters.

The local parameter sensitivities with respect to some parameter p_k are defined as

$$\mathbf{s}_k(t, \mathbf{p}) := \left. \frac{\partial \mathbf{x}(t, \mathbf{p})}{\partial p_k} \right|_{\mathbf{p}=\mathbf{p}_0} \tag{2}$$

which is the vector of the derivatives of all variables x_i with respect to the parameter p_k. Similar to the dynamics in Eq. (1), the parameter sensitivities' time evolution follows a system of ODEs given by differentiating Eq. (2) with respect to t:

$$\dot{\mathbf{s}}_k(t, \mathbf{p}) := \frac{\partial^2 \mathbf{x}(t, \mathbf{p})}{\partial p_k \partial t} = \underbrace{\frac{\partial \mathbf{f}(\mathbf{x}(t, \mathbf{p}), \mathbf{p})}{\partial \mathbf{x}(t, \mathbf{p})}}_{:=\mathbf{J}_f(\mathbf{x}(t), \mathbf{p})} \mathbf{s}_k(t, \mathbf{p}) + \frac{\partial \mathbf{f}(t, \mathbf{p})}{\partial p_k}, \tag{3}$$

where $\mathbf{J}_f(\mathbf{x}(t))$ is the Jacobian matrix of $\mathbf{f}(\mathbf{x}(t))$ with respect to $\mathbf{x}(t)$; note that we drop explicit dependencies on $\mathbf{p}$ to simplify notation. Initial conditions for Eq. (3) are set according to whether the initial conditions for the states in Eq. (1) depend on the parameters or not [3].

Consider, for example, the biochemical scheme of a Michaelis-Menten type enzymatic reaction

$$x_1 + x_2 \underset{k_2}{\overset{k_1}{\rightleftarrows}} x_3 \xrightarrow{k_3} x_1 + x_4 \tag{4}$$

where x_{1-4} correspond to enzyme, substrate, enzyme-substrate complex, and product concentrations, respectively. With mass-action kinetics, the reaction network translates to the dynamic system

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} -k_1 x_1 x_2 + (k_2 + k_3) x_3 \\ -k_1 x_1 x_2 + k_2 x_3 \\ +k_1 x_1 x_2 - (k_2 + k_3) x_3 \\ +k_3 x_3 \end{pmatrix}, \quad \mathbf{p} = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix}.$$

Such problems are often well solved by general purpose ODE solvers, but (bio)chemical reaction networks offer a number of features that may be exploited by more specialized solvers, resulting in faster and/or more precise simulations. For instance, in enzyme kinetics, reversible association and dissociation processes are usually much

faster than product formation. The resulting stiffness severely limits the types of numerical methods that can be used for ODE integration.

An opportunity for increasing solver efficiency, however, presents itself because most (bio)chemical reaction networks are only weakly interconnected. More specifically, the change in every concentration x_i usually depends on the concentration of very few other products. Poor connectivity is reflected in *sparse* Jacobians $\mathbf{J}_f(\mathbf{x}(t))$, where non-zero elements correspond to interactions between components (that is, we have a correspondence to the network graph's adjacency matrix). For the simple example Eq. (4),

$$\mathbf{J}_f(\mathbf{x}(t)) = \begin{pmatrix} \bullet & \bullet & \bullet & \circ \\ \bullet & \bullet & \bullet & \circ \\ \bullet & \bullet & \bullet & \circ \\ \circ & \circ & \bullet & \circ \end{pmatrix},$$

with closed and open circles indicating non-zero and zero elements, respectively. Even in this dense sub-network, the number of non-zeros $nnz_J = 5/8n_x^2$ implies that we do not need to compute a substantial number of terms to determine the Jacobian.

Many large-scale biological networks have a scale-free structure, that is, most of their nodes have few interactions, but a small number of *hubs* with many interactions exist [14]. This prevents an easy decomposition of a large network into subsystems that can be handled (and integrated) independently. Therefore, despite the sparsity of the Jacobian, model size remains a major issue for numerical performance.

Two more general aspects also need to be considered. Firstly, due to the growing use of abstract modeling software, the reactions and the underlying reaction equations are usually available to us as abstract models, such as SMBL [15], which we can analyze and manipulate analytically. Secondly, since parametric uncertainty is abundant in biology, sensitivity analysis, *i.e.* the integration of the parameter sensitivities $\mathbf{s}_k(t)$, requires particular attention. Hence, an ideal ODE solver for our application domain would efficiently and reliably handle large, stiff dynamic systems including their parameter sensitivities, and optimally exploit the systems' non-trivial sparsity and analytic access.

Methods for ODE integration

Almost all ODE integrators work under the assumption that the change in each variable x_i over time can be modeled using a polynomial in t. Consider the Taylor expansion of the variables $\mathbf{x}(t)$ around $t = t_0$ to advance the system by a step of size h:

$$\mathbf{x}(t_0+h) = \mathbf{x}(t_0)+h\frac{\partial \mathbf{x}}{\partial t}(t_0)+\frac{h^2}{2!}\frac{\partial^2 \mathbf{x}}{\partial t^2}(t_0)+\frac{h^3}{3!}\frac{\partial^3 \mathbf{x}}{\partial t^3}(t_0)+\dots \tag{5}$$

If the factors $\partial^k \mathbf{x}(t)/\partial t^k/k!$ decrease sufficiently quickly and the higher-order terms become insignificant as of some degree n, then we can reliably approximate the new solution by a polynomial of degree n in h. In *explicit* integrators, previously computed values of $\mathbf{x}(t)$ and the derivatives $\partial \mathbf{x}(t)/\partial t$ are used to construct a polynomial $\mathbf{g}_n(t)$ of degree n and to extrapolate the value of $\mathbf{x}(t+h) \approx \mathbf{g}_n(t+h)$. In *implicit* integrators, a solution $\mathbf{x}(t+h)$ is sought such that it matches that of a polynomial $\mathbf{g}_n(t)$ of degree n interpolated through previous values of $\mathbf{x}(t)$ and/or their derivatives, and the derivative at the solution $\mathbf{x}(t+h)$ itself. In general, implicit integrators are more accurate for stiff ODEs, where the derivatives in Eq. (5) do not decay sufficiently quickly.

Within the two larger classes, different integrators are characterized by the amount of previous values of $\mathbf{x}(t)$ and their derivatives which they use to approximate $\mathbf{x}(t+h)$. Table 1 lists some common integration methods; see [16] for a comprehensive review.

Despite the commensurate degree of freedom in designing ODE integrators, and the number of algorithms for the numerical integration of ODEs that have been published over the past 40 years, only very few of them have found wide-spread application. Practical considerations—any method should be easily accessible to its end users, who are usually not interested in manipulating or even formulating the underlying equations themselves—are certainly major causes for this convergence [17]. However, a closer analysis of the most popular solvers for stiff ODE systems reveals another cause, namely incremental evolution.

In this area, the first major piece of software was the `GEAR` package [18], which by 1996 evolved into `cvode`

Table 1 Common ODE integration schemes and the values that are used to approximate the polynomial in Eq. (5)

Integrator	Nodes, explicit[1]	Nodes, implicit
Euler	$\mathbf{x}(t), \dot{\mathbf{x}}(t)$	$\mathbf{x}(t), \dot{\mathbf{x}}(t+h)$
Backward Differentiation Formula (BDF)	$\dot{\mathbf{x}}(t), \mathbf{x}(t-h_k)$	$\mathbf{x}(t), \dot{\mathbf{x}}(t+h), \mathbf{x}(t-h_k)$
Adams-Moulton (AM)	$\dot{\mathbf{x}}(t), \dot{\mathbf{x}}(t-h_k)$	$\mathbf{x}(t), \dot{\mathbf{x}}(t+h), \dot{\mathbf{x}}(t-h_k)$
Second-derivative rule (this work)	—	$\mathbf{x}(t), \dot{\mathbf{x}}(t), \ddot{\mathbf{x}}(t), \dot{\mathbf{x}}(t+h), \ddot{\mathbf{x}}(t+h)$

[1] For multi-step methods, the index $k = 0, -1, \dots, -n$.

[19], a part of the Sundials suite of nonlinear and differential/algebraic equation solvers [20]. The default integrators in Matlab (The MathWorks, Natick, MA) such as `ode15s` [21] employ similar integration rules and error estimates. Both the Sundials suite and Matlab are used increasingly in systems biology [22], but it is not evident that they are optimal for this application domain.

Methods

A second-derivative integrator

All ODE solvers mentioned above use only values of $\mathbf{x}(t)$ and $\dot{\mathbf{x}}(t)$ to approximate $\mathbf{x}(t+h)$. Here, we differ from these methods in that we also employ the second derivatives:

$$\ddot{\mathbf{x}}(t) := \frac{\partial^2 \mathbf{x}(t)}{\partial t^2} = \frac{\partial \mathbf{f}(\mathbf{x}(t))}{\partial t} = \mathbf{J}_f(\mathbf{x}(t))\mathbf{f}(\mathbf{x}(t)) \tag{6}$$

(for notational simplicity, we will write $\mathbf{J}_f(t)$ and $\mathbf{f}(t)$ instead of $\mathbf{J}_f(\mathbf{x}(t))$ and $\mathbf{f}(\mathbf{x}(t))$, respectively). Note that this second derivative with respect to the time t should not be confused with the second-order sensitivities described and used in [9,10,23], which are the second derivatives of the system variables with respect to the system parameters.

The use of second derivatives was first suggested in [24] and later studied in detail in [25], [26] and [27], and the resulting formulas were shown to have good stability properties. A more recent study [28] reinforces the stability and potential efficiency gains for stiff systems through second-derivative methods. However, despite several published implementations [27,28], these methods have not yet found wide acceptance because, despite being A-stable, they are only stable at infinity if only the second derivative at $t+h$ is used [25] (see Section S3 in Additional file 1 and Additional file 2 for details).

The second derivatives in Eq. (6) may seem somewhat clumsy and expensive to evaluate since they require the construction of the Jacobian $\mathbf{J}_f(t)$ and the evaluation of a matrix-vector multiplication $\mathbf{J}_f(t)\mathbf{f}(t)$. Remember, however, that (bio)chemical reaction network models typically have sparse Jacobians. As a consequence, the cost of constructing $\mathbf{J}_f(t)$ and of evaluating the product $\mathbf{J}_f(t)\mathbf{f}(t)$ grows only linearly with the number of variables and not quadratically, as the matrix-vector product would imply. Furthermore, since we usually have an abstract representation of the governing equations, we can compute each entry of $\ddot{\mathbf{x}}(t)$ explicitly, much in the same way we evaluate the entries of $\mathbf{f}(t)$. For instance, the explicit second derivatives of the system Eq. (4) are:

$$\ddot{\mathbf{x}}(t) = \mathbf{J}_f(t)\mathbf{f}(t) = \begin{pmatrix} -k_1x_2f_1 - k_1x_1f_2 + (k_2+k_3)f_3 \\ -k_1x_2f_1 - k_1x_1f_2 + k_2f_3 \\ k_1x_2f_1 + k_1x_1f_2 - (k_2+k_3)f_3 \\ k_3f_3 \end{pmatrix}.$$

In most cases, the evaluation of the second derivatives is not much more expensive than the evaluation of $\mathbf{f}(t)$.

For our second-derivative integrator, we construct an interpolating polynomial $\mathbf{g}_4(t)$ of degree n=4 matching $\mathbf{x}(t)$ and the first and second derivatives at times t and $t+h$. This implicit method requires that we find $\mathbf{x}(t+h)$ such that

$$\mathbf{x}(t+h) = \mathbf{g}_4(t+h)$$

which, expanding $\mathbf{g}_4(t+h)$, gives us

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \frac{h}{2}\left[\dot{\mathbf{x}}(t) + \dot{\mathbf{x}}(t+h)\right] + \frac{h^2}{12}\left[\ddot{\mathbf{x}}(t) - \ddot{\mathbf{x}}(t+h)\right]. \tag{7}$$

where the right-hand side is the polynomial through $\mathbf{x}(t)$, $\dot{\mathbf{x}}(t)$, $\ddot{\mathbf{x}}(t)$, $\dot{\mathbf{x}}(t+h)$ and $\ddot{\mathbf{x}}(t+h)$ evaluated at $t+h$ (this is, incidentally, the original scheme proposed in [24]). The solution to this system of equations can be computed iteratively. More specifically, we start from an initial guess $\tilde{\mathbf{x}}(t+h)$ that is computed with an explicit formula, and use a simplified Newton's Method:

$$\mathbf{x}(t+h) \leftarrow \mathbf{x}(t+h) - [\mathbf{M}(t+h)]^{-1}\left(\mathbf{g}_4(t+h) - \mathbf{x}(t+h)\right),$$

$$\mathbf{M}(t+h) := \frac{h}{2}\tilde{\mathbf{J}}_f(t+h) - \frac{h^2}{12}\tilde{\mathbf{J}}_{Jf}(t+h) - \mathbf{I}, \tag{8}$$

where $\mathbf{M}(t+h)$ is the Newton iteration matrix and $\mathbf{J}_{Jf}(t)$ is the Jacobian of Eq. (6) with respect to $\mathbf{x}(t)$:

$$\mathbf{J}_{Jf}(t) := \frac{\partial \mathbf{J}_f(t)\mathbf{f}(t)}{\partial x} = \frac{\partial \mathbf{J}_f(t)}{\partial x}\mathbf{f}(t) + \left(\mathbf{J}_f(t)\right)^2 \tag{9}$$

The Jacobians $\tilde{\mathbf{J}}_f(t+h)$ and $\tilde{\mathbf{J}}_{Jf}(t+h)$ are evaluated at the initial guess $\tilde{\mathbf{x}}(t+h)$.

Using a second-derivative scheme, the evaluation of each Newton iteration is roughly twice as expensive as for first-derivative methods of the same degree since, in addition to $\mathbf{f}(t+h)$, we must also evaluate $\mathbf{J}_f(t+h)\mathbf{f}(t+h)$. The advantage of this scheme, however, becomes obvious once we consider the truncation error. By replacing $\mathbf{x}(t+h)$ with the Taylor expansion around t, we obtain

$$\mathbf{g}_4(t_{n+1}) - \mathbf{x}(t_{n+1}) \approx \frac{1}{720}h^5\mathbf{x}^{(5)}(\xi), \quad \xi \in [t_n, t_{n+1}] \tag{10}$$

for the truncation error of our second-derivative formula. For first-derivative methods of the same degree, assuming a constant step size h, this error is

$$\mathbf{BDF}_4(t_{n+1}) - \mathbf{x}(t_{n+1}) \approx \frac{72}{750}h^5\mathbf{x}^{(5)}(\xi), \quad \xi \in [t_{n-3}, t_{n+1}],$$

$$\mathbf{AM}_4(t_{n+1}) - \mathbf{x}(t_{n+1}) \approx \frac{19}{720}h^5\mathbf{x}^{(5)}(\xi), \quad \xi \in [t_{n-3}, t_{n+1}],$$

in the case of the BDF and the Adams-Moulton formula of degree four, respectively. These truncation errors are 72 times and 19 times larger than the error of our second-derivative formula (assuming the fifth derivative $\mathbf{x}^{(5)}(t)$ is

approximately constant in $[t_{n-3}, t_{n+1}]$, see Section S2 and Figure S1A in Additional file 1 and Additional file 2 for details). The large difference stems from the dependence of the interpolation error on the width of the interpolation interval, *e.g.* for the BDF and the Adams-Moulton formula, this interval is four times larger.

Error estimates and step size adjustment

In any ODE integration scheme, the local error estimate and the step-size adjustment are crucial to both its accuracy and its efficiency. The step-size adjustment uses the error estimate of a previous integration step to predict the largest possible next step h satisfying the required tolerance. With imprecise error estimates, the step-size adjustment has to be conservative to preserve accuracy, or it risks producing an imprecise result.

In most implicit ODE solvers, the local error is either estimated from the difference between the initial estimate $\tilde{\mathbf{x}}(t+h)$, usually computed with an explicit rule, and the final converged step $\mathbf{x}(t+h)$, or as the difference between two rules of different degree over the previous $\mathbf{x}(t)$ and the converged step $\mathbf{x}(t+h)$. These approaches mainly consider computational efficiency because, ideally, to estimate the error of a formula of degree d_1, we need to compute a better approximation of degree $d_2 > d_1$. The difference between both converged solutions $\mathbf{x}_1(t+h)$ and $\mathbf{x}_2(t+h)$ can then be used to approximate the difference between the lower-degree estimate and the exact solution $\mathbf{x}^\star(t+h)$. However, this requires *two* Newton iterations to compute both solutions and if both rules have different weights for the values of $\dot{\mathbf{x}}(t+h)$ and $\ddot{\mathbf{x}}(t+h)$, we would need to invert or decompose two different matrices to compute a Newton iteration (Eq. (8)).

We propose a different approach that may better reconcile accuracy with computational cost. We first compute the converged lower-degree solution $\mathbf{x}_1(t+h)$ and use it as an initial estimate for the Newton iteration of the higher-degree solution. Since we are not actually interested in the exact solution $\mathbf{x}_2(t+h)$, but only in an approximation of the difference between the two solutions, it suffices to compute just one Newton step to get a first-oder approximation of that difference. Note that, in principle, this still requires the inversion or decomposition of a different matrix for the Newton iteration.

However, for our second-derivative solver, we can compute the second approximation as the polynomial $\mathbf{g}_5(t)$ that interpolates $x(t)$ at the same nodes as $\mathbf{g}_4(t)$ plus the second to last node $\mathbf{x}(t-h_{-1})$. In this case, the weights in the Newton iteration matrix are similar. If the current step size h and the previous step size h_{-1} are equal, the weights for $\mathbf{J}_f(t+h)$ and $\mathbf{J}_{ff}(t+h)$ are 14/31 and 2/31, respectively, which is close to the values for $\mathbf{g}_4(t)$ of 1/2 and 1/12. We therefore re-use the Newton iteration matrix in Eq. (8) to compute the first approximation $\mathbf{x}_1(t+h)$ and obtain the local error estimate

$$\varepsilon := [\mathbf{M}(t+h)]^{-1}\left(\mathbf{g}_5(t+h) - \mathbf{x}(t+h)\right). \tag{11}$$

Note that the estimate ε approximates the truncation error Eq. (10). Assuming that $\mathbf{x}^{(5)}(t)$ varies only slowly between two time steps, we can compute a scaling σ such that the error of the next step of size σh is equal to a prescribed tolerance τ:

$$\varepsilon = \frac{h^5}{720}\mathbf{x}^{(5)}(\xi),\ \tau = \frac{(\sigma h)^5}{720}\mathbf{x}^{(5)}(\xi) \quad\Longrightarrow\quad \sigma = \left(\frac{\tau}{\varepsilon}\right)^{1/5}. \tag{12}$$

Note that if the assumptions on $\mathbf{x}^{(5)}(t)$ do not hold, the error estimate in the next time step will fail, causing the step size to be reduced automatically. Furthermore, if we adjust h to fulfill the requested tolerance τ exactly, the error estimate will be larger than τ approximately half of the time. Therefore, in practice, we choose σ such that the next error will be $\tau/2$. This gives us a recipe to adjust the step size from one integration step to the next and, hence, the last key ingredient of a functional second-derivative ODE solver (see Section S2 and Figure S1B of the Additional file 1 and Additional file 2 for details).

Parameter sensitivities

For n_x variables and n_p parameters, the naive approach to sensitivity calculation implies integrating a system of $n_x \times (1+n_p)$ variables and, by consequence, inverting or decomposing matrices of that size within the Newton iteration. However, the system variables $\mathbf{x}(t)$ do not depend on the parameter sensitivities, yet the sensitivities depend on $\mathbf{x}(t)$. Hence, we can, in each step, first compute the values $\mathbf{x}(t+h)$ and, once they have converged, compute the $\mathbf{s}_k(t+h)$ in a separate step using the same integration rule. This *staggered* approach was first introduced in Caracotsis & Stewart [29], then extended by Maly & Petzold [30], and finally implemented in the Sundials `cvodes` ODE solver, a modified version of `cvode` capable of sensitivity analysis [31].

To integrate the parameter sensitivities in our second-derivative solver in a similar way, we need to compute the second derivatives

$$\ddot{\mathbf{s}}_k(t) := \frac{\partial^2 \mathbf{s}_k(t)}{\partial t^2} = \frac{\partial}{\partial t}\left(\mathbf{J}_f(t)\mathbf{s}_k(t) + \frac{\partial \mathbf{f}(t)}{\partial p_k}\right) = \mathbf{J}_{ff}\mathbf{s}_k(t) + \frac{\partial \ddot{\mathbf{x}}(t)}{\partial p_k}. \tag{13}$$

The equation in the implicit step using the second-derivative rule in Eq. (7) for the parameter sensitivities $\mathbf{s}_k(t)$ thus becomes

$$\mathbf{s}_k(t+h) = \mathbf{s}_k(t) + \frac{h}{2} \times \left(\dot{\mathbf{s}}_k(t) + \mathbf{J}_f(t+h)\mathbf{s}_k(t+h) + \frac{\partial \mathbf{f}(t+h)}{\partial p_k} \right) + \frac{h^2}{12} \left(\ddot{\mathbf{s}}_k(t) - \mathbf{J}_{ff}(t+h)\mathbf{s}_k(t+h) - \frac{\partial \ddot{\mathbf{x}}(t+h)}{\partial p_k} \right), \quad (14)$$

which, after isolating the sole unknown term $\mathbf{s}_k(t+h)$ leads to

$$\left[\mathbf{I} - \frac{h}{2}\mathbf{J}_f(t+h) + \frac{h^2}{12}\mathbf{J}_{ff}(t+h) \right] \mathbf{s}_k(t+h) = \mathbf{s}_k(t) + \frac{h}{2} \left(\dot{\mathbf{s}}_k(t) + \frac{\partial \mathbf{f}(t+h)}{\partial p_k} \right) + \frac{h^2}{12} \left(\ddot{\mathbf{s}}_k(t) - \frac{\partial \ddot{\mathbf{x}}(t+h)}{\partial p_k} \right). \quad (15)$$

We then have two alternatives to compute $\mathbf{s}_k(t+h)$: either iteratively using Newton's method to solve Eq. (14), or directly by inverting or decomposing the matrix on the left-hand side of Eq. (15). The iterative approach via Eq. (14) is equivalent to the one suggested in [30], and we can re-use the inverted or decomposed iteration matrix used to compute the variables $\mathbf{x}(t+h)$ in Eq. (8). However, one has to re-evaluate the Jacobians at each iteration to compute $\dot{\mathbf{s}}_k(t)$ and $\ddot{\mathbf{s}}_k(t)$ because $\tilde{\mathbf{J}}_f(t+h)$ and $\tilde{\mathbf{J}}_{ff}(t+h)$ are evaluated at the initial estimate $\tilde{\mathbf{x}}(t+h)$, and not at the converged solution $\mathbf{x}(t+h)$. To compute $\mathbf{s}_k(t+h)$ directly, which corresponds to the original approach in [29], we need to re-compute the Jacobians and the inverse or decomposition of the left-hand side of Eq. (15). For small n_x, however, this extra matrix computation may be advantageous over running the Newton iteration for each parameter p_k.

Furthermore, if the Jacobians do not vary significantly over time, they can be re-used as the matrices $\tilde{\mathbf{J}}_f(t)$ and $\tilde{\mathbf{J}}_{ff}(t)$ for the Newton iteration of the *next* step. Such an approach offers an advantage if the cost of running an additional n_p Newton iterations to compute the parameter sensitivities iteratively outweighs the cost incurred by the slower convergence due to using older Jacobians in the next step. In our second-derivative integrator, we therefore compute the parameter sensitivities directly as per Eq. (15).

Framework for conversion of SBML models

In order to generate the matrices $\mathbf{J}_f(t)$ and $\mathbf{J}_{ff}(t)$, as well as the second derivative $\ddot{\mathbf{x}}(t)$ automatically, we established a framework that automatically translates arbitrary models from the standard SBML format [15] to Matlab functions or C-language code. The framework also generates routines to compute the parameter derivatives $\partial \mathbf{f}(t)/\partial \mathbf{p}$ and $\partial \ddot{\mathbf{x}}(t)/\partial \mathbf{p}$ necessary for the parameter sensitivity computations. This conversion, which needs to be done only once per model, exploits the sparsity of the corresponding matrices by generating compact expressions for their non-zero entries only, making them efficient to evaluate. It uses the Matlab Symbolic Toolbox to manipulate, differentiate and simplify the resulting expressions automatically (see Sections S1.3 and S1.4 in the Additional file 1 and Additional file 2 for details).

Results and discussion

Implementation and testing

We implemented the second-derivative ODE integrator as `odeSD` in Matlab and as $\texttt{odeSD}_{\texttt{mex}}$ in the C programming language, using the Matlab `mex` interface with calls to the LAPACK and BLAS libraries for the linear algebra operations. Both solvers provide an interface similar to that of the Matlab default integrators. Additionally, a native C-language version, $\texttt{odeSD}_{\texttt{c}}$, was implemented for use outside of the Matlab programming environment. All implementations could operate on any type of ODE-based model, but the overall implementation is targeted to systems biology models in standard SBML format, for which we developed an automatic model conversion framework (see Methods). The implementation details are described in Sections S1.1 and S1.2 of the Additional file 1 and Additional file 2.

We compared our algorithm against three integrators which use Newton's method to compute each implicit step:

1. Matlab's default integrator for stiff systems, `ode15s` [21], which uses a 5-point Numerical Differentiation Formula (NDF), a more stable variant of the BDF integration rules; it is used as the default integrator in SBToolbox2 [22],
2. the `cvode` integrator from the Sundials suite [20] which employs variable-order BDFs of up to degree 4; it is the integrator used in the SBML ODE Library (SOSlib) [32], and
3. the `radau5` integrator, a fifth-order three-stage implicit Runge-Kutta method for stiff systems described in [33] and implemented in Matlab [34].

The Matlab interface supplied by the `sundialsTB` toolbox [35] served to run the Sundials integrators which are implemented in C.

For performance evaluation, we selected a number of curated systems biology models from the BioModels database [36] (Table 2). This set comprises systems of different sizes (up to the largest models available in the database) and characteristics, namely convergence to steady-state and (stiff) oscillatory behavior. Note that all models have sparse Jacobians, as is evident from the number of non-zero elements $nnz(J_f)$.

Table 2 Systems biology test models and their key characteristics, namely number of states (n_x), number of parameters (n_p), number of non-zeros of the Jacobians $nnz(\mathrm{J}_f)$, integration time interval (t), and biological system described by the model

Model	n_x	n_p	$nnz(\mathrm{J}_f)$	$nnz(\mathrm{J}_{jf})$	t	Comments
Hornberg *et al.* [37]	8	19	22	32	[0,100]	Steady state, ERK[1] phosphorylation and kinase/phosphatase control.
Kholodenko *et al.* [38]	23	51	137	372	[0,100]	Steady state, short term signaling by the EGF receptor.
Singh *et al.* [39]	66	109	323	846	[0,35 000]	Steady state, IL-6 signal transduction in hepatocytes.
Borisov *et al.* [40]	90	136	468	2156	[0,1 000]	Steady state, insulin-EGF network interactions in mitogenic signaling.
Ung *et al.* [41]	200	314	956	4038	[0,4 000]	Steady state, regulation of EGFR endocytosis and EGFR-ERK signaling by crosstalk.
Elowitz & Leibler [42]	6	8	12	18	[0,1 000]	Oscillatory, synthetic network of transcriptional regulators.
Leloup & Goldbeter [43]	10	48	30	52	[0,300]	Oscillatory, circadian oscillations of PER and TIM proteins in *Drosophila*.
Wolf *et al.* [44]	13	40	47	99	[0,300]	Oscillatory, autonomous metabolic oscillations in continuous culture of *S. cerevisiae*.
Goldbeter & Pourquié [45]	20	73	47	76	[0,300]	Oscillatory, segmentation clock by crosstalk of Notch, Wnt, and FGF pathways.
Xie & Kulasiri [46]	24	49	57	107	[0,100]	Oscillatory, circadian rhythms in *Drosophila* with interlocked feedback loops.

[1] Abbreviations used: ERK = extracellular regulated kinase; EGF = epidermal growth factor; IL-6 = interleukin 6; FGF = fibroblast growth factor.

Integrator performance without parameter sensitivities

The results of the performance comparison without sensitivity analysis for a wide range of integration tolerances are summarized in Figure 1 (see Additional file 1: Figures S2-3 for details). The average computational times for our integrator were comparable (`odeSD`$_{\text{mex}}$ vs. `cvode`) to, or slightly lower (`odeSD` vs. `ode15s` or `radau5`) than those of the first-derivative solvers (Figure 1A), except for low numerical tolerances. Importantly, the second-derivative integrator required approximately half as many steps as `ode15s` or `cvode` (Figure 1B), despite these three integrators using rules of the same degree of precision. The `radau5` integrator used less steps than `odeSD`, but it computes two additional intermediate steps per full step. The smaller number of steps in `odeSD` is due to a combination of both the smaller truncation error of the second-derivative rule and the better accuracy of the improved error estimate. For more detailed results and discussion, see Section S3 and Figure S2 of the the Additional file 1 and Additional file 2.

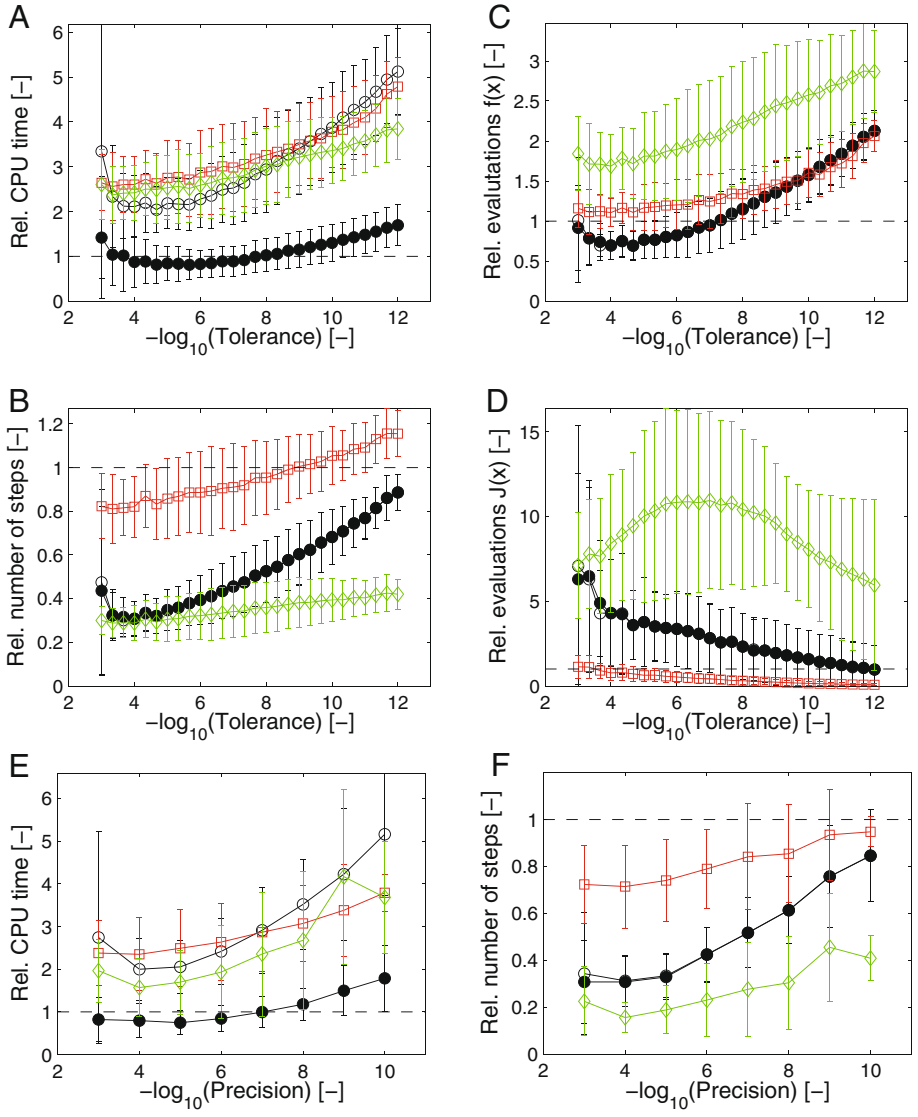

Figure 1 Performance comparison without parameter sensitivities. Performance comparison for integration of ODE-based systems biology models without parameter sensitivities. (**A**) Computation times, (**B**) number of integration steps, (**C**) number of r.h.s. evaluations $\mathbf{f}(\mathbf{x})$, and (**D**) number of evaluations of the Jacobian $\mathbf{J}_f(\mathbf{x})$ as a function of the relative numerical tolerance. Symbols specify the integrators `ode15s` (open red squares), `radau5` (open green diamonds), `odeSD` (open black circles), and `odeSD`$_{\text{mex}}$ (filled black circles), respectively. Performance metrics are normalized to the corresponding measures for `cvode` and averaged (mean ± std.) over all models, which were integrated over the time spans given in Table 2; the dashed line indicates performance equal to `cvode`. (**E**) Computation times and (**F**) number of integration steps as a function of numerical precision (see main text for definition) in analogy to (**A**) and (**B**).

To assess the relative accuracy of `odeSD`, we compared the results of all models computed with different relative tolerances with an 'accurate' reference solution computed using `radau5` with the relative tolerance set to 10^{-15}, analogously to the precision/work tests in [26]. The measured precision for each model and integrator is the maximum relative error in the final step for each state larger than machine precision in the reference solution. Figure 2 shows these results for all models in Table 2. These results are summarized in Figure 1E with the CPU time averaged over all models and the precision binned to the closest power of 10. Overall, without computing the parameter sensitivities, the new integrator is competitive, in terms of accuracy and efficiency, with highly optimized state of the art solvers for hard numerical problems in our application domain.

Integrator performance with parameter sensitivities

The performance comparison with sensitivity calculations requires two additional considerations: Since `ode15s` and `radau5` do not provide any special functionality for computing parameter sensitivities, we used an augmented system of size $n_x \times (n_p + 1)$ including an analytic sparse Jacobian for each model, and the sensitivities $\mathbf{s}_k(t)$, $k = 1 \ldots m$ were integrated alongside the system variables. `cvodes`, the sensitivity analysis-enabled version of `cvode` from the SUNDIALS package, uses a simultaneous integrator based on the method of Maly & Petzold [47]. Optionally, the staggered integrator of Feehery *et al.* [30] can be selected, but this did not produce better results.

In all cases, parameter sensitivities were integrated to the same precision as the system variables. As with the integration without sensitivities, precision/work diagrams were computed for all models with sensitivities, omitting the cases in which `ode15s` failed completely.

As one detailed example, Figure 3A shows the computation times for a relative tolerance of 10^{-6}. In most cases, the compute times with sensitivities are substantial (see also Additional file 1: Figure S3 for other tolerances). For `ode15s` and `radau5`, the size of the augmented system quickly becomes a problem as the solution of the linear system of equations in the Newton iteration scales cubically with the number of variables. In terms of the additional effort for the sensitivity computation, we note that the second-derivative integrators are more efficient, often increasing the compute time only

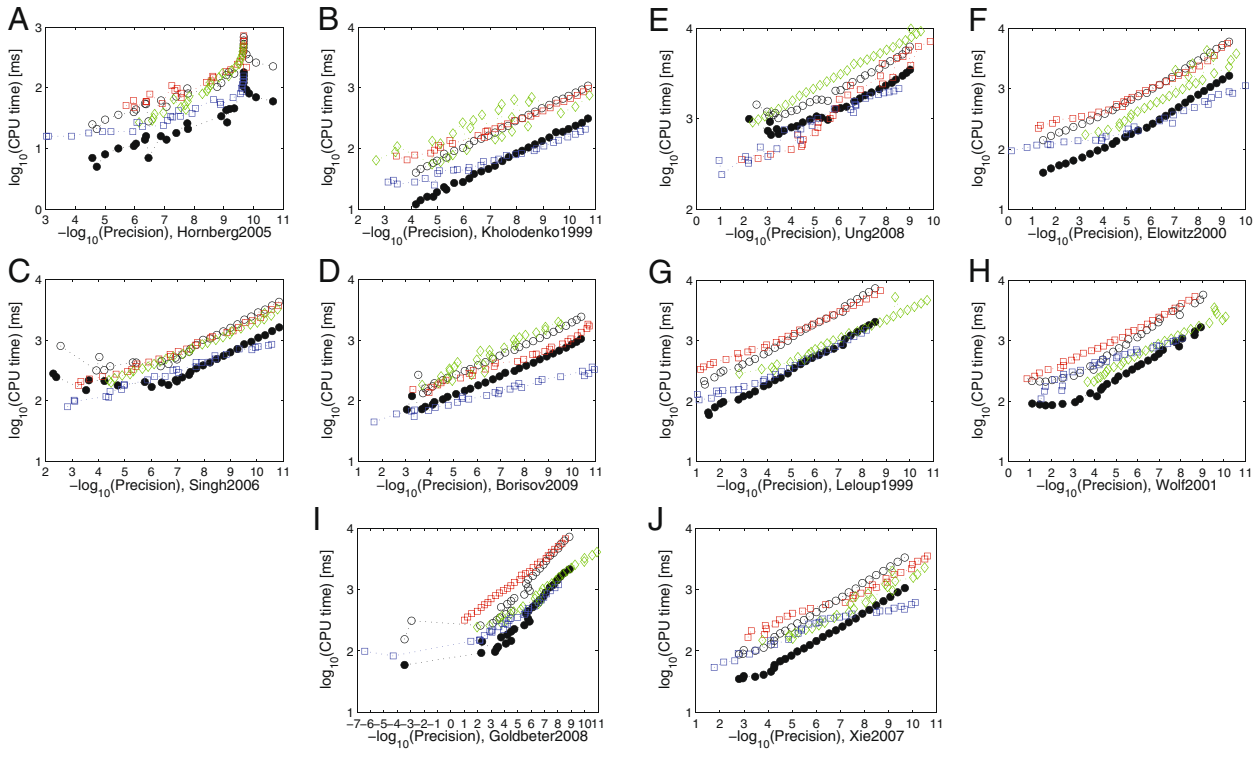

Figure 2 Precision/work diagrams without parameter sensitivities. Precision-work diagrams for integration without parameter sensitivities. (**A-J**) Computation times for the individual models (see X-axis for model specifications) as a function of precision using `odeSD` (open black circles), $\texttt{odeSD}_{\texttt{mex}}$ (filled black circles), `ode15s` (red squares), `radau5` (green diamonds), and `cvodes` (blue squares). All models were integrated for the time spans shown in Table 2.

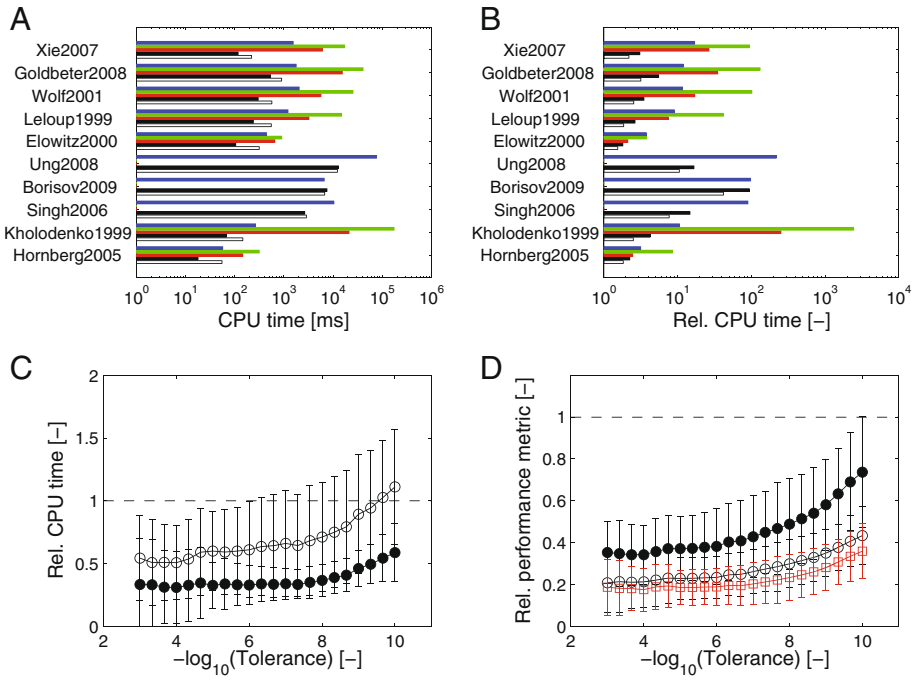

Figure 3 Performance comparison with parameter sensitivities. Performance comparison with parameter sensitivities. (**A**) Computation times for the individual models listed in Table 2 with relative tolerance of 10^{-6} using `odeSD` (white bars), $\texttt{odeSD}_{\texttt{mex}}$ (black), `ode15s` (red), `radau5` (green), and `cvodes` (blue). Due to the explosion in compute time, the three largest steady-state models were not evaluated with `ode15s` and `radau5`. (**B**) CPU times with sensitivity calculation as in (**A**) relative to CPU times without sensitivity calculation. (**C**) Average, normalized (see below) CPU times with sensitivities as a function of the relative numerical tolerance for `odeSD` (open circles) and $\texttt{odeSD}_{\texttt{mex}}$ (filled circles) relative to `cvodes`. (**D**) Relative numbers of integration steps (open black circles), of function evaluations $\mathbf{f}(\mathbf{x})$ (filled black circles), and of evaluations of the Jacobians $\mathbf{J}.(\cdot)$ (open red squares) for $\texttt{odeSD}_{\texttt{mex}}$ compared to `cvodes`, respectively. In all cases, model and sensitivity equations were integrated for the time spans shown in Table 2. Performance metrics in (**C**, **D**) are normalized to the corresponding measures for `cvodes` and averaged (mean ± std.); the dashed line indicates performance equal to `cvodes`.

two- or three-fold, whereas the overhead is substantial for `cvodes` (Figure 3B). The results for `cvodes` are best explained if we keep in mind that whenever the right-hand side $\mathbf{f}(\cdot)$ is evaluated, the algorithm also computes $\dot{\mathbf{s}}_k(t)$, $k = 1 \ldots n_p$, which, as per Eq. (3), requires an evaluation of the Jacobian $\mathbf{J}_f(\cdot)$. As a consequence, our second-derivative integrator outperforms the Sundials solver when implemented in C using the Matlab `mex` interface ($\texttt{odeSD}_{\texttt{mex}}$), and even in native Matlab (`odeSD`) for all larger models. Note that the higher compute times for $\texttt{odeSD}_{\texttt{mex}}$ vs. `odeSD` in some cases are the result of a more refined handling of near-singular matrices by Matlab in the latter.

In order to obtain results independent of any potential inefficiencies of the Matlab interface, the same performance analysis was run using $\texttt{odeSD}_{\texttt{c}}$ and `cvodes` with natively compiled C-language functions for the right-hand sides. The results of this comparison are summarized in Figure 4 (see Additional file 1: Figure S4 in the Additional file 1 and Additional file 2 for the detailed precision-work diagrams).

The higher efficiency of `odeSD`, $\texttt{odeSD}_{\texttt{mex}}$ and $\texttt{odeSD}_{\texttt{c}}$ holds also for averages over all models and for a wide range of numerical tolerances (Figure 3C). Except for high-precision integration, we achieve approximately two to three-fold speed-ups. These general findings also hold when compute time is assessed as a function of numerical precision (Figure 5).

To explain the performance, consider that although `odeSD` and $\texttt{odeSD}_{\texttt{mex}}$ also evaluate the Jacobians in each step, both the much smaller number of steps required (Figure 3D), which is of no particular advantage when the parameter sensitivities are not computed, and the re-use of the Jacobians for sensitivity computations lead to significantly shorter execution times since substantially fewer evaluations of $\mathbf{f}(\cdot)$ and of the Jacobians $\mathbf{J}_{(\cdot)}(\cdot)$ are needed (Figure 3D). Since the latter dominates the integration cost, all three versions of `odeSD` outperform `cvodes` in all but the smallest systems. These results are not a consequence of the sparsity of the systems *per se*, but of the more precise integration rule which can be computed efficiently thanks to sparsity. The better error estimate

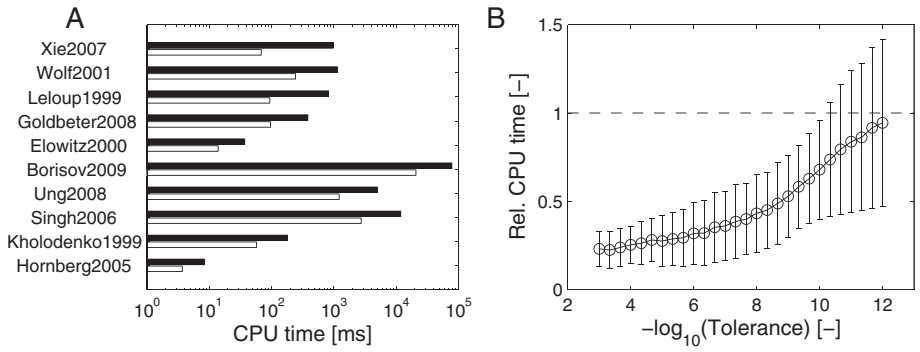

Figure 4 Performance comparison of C-language integrators with parameter sensitivities. Performance comparison with parameter sensitivities of the C-language version $\texttt{odeSD}_\texttt{c}$ with `cvodes` using the automatically generated, compiled C-language right-hand side and Jacobian functions. (**A**) Computation times for the individual models listed in Table 2 with relative tolerance of 10^{-6} using `odeSD` (white bars) and `cvodes` (black). (**B**) Average, normalized (see below) CPU times as a function of the relative numerical tolerance for `odeSD` relative to `cvodes`. In all cases, model and sensitivity equations were integrated for the time spans shown in Table 2. Performance metrics in (**B**) are normalized to the corresponding measures for `cvodes` and averaged (mean ± std.); the dashed line indicates performance equal to `cvodes`.

and the computation of local parametric sensitivities are therefore particular strengths of our ODE solver based on second derivatives.

Conclusions

We have presented an integrator for ODE systems resulting from the modeling of chemical and biological reaction networks, which are often stiff and sparse. For the realistic systems biology models tested, the new integrator outperforms commonly used state of the art integrators when parameter sensitivities are required. It is competitive in integrating the system equations alone, despite limitations for specific models near the steady state. The improvements with respect to sensitivity calculations are critical for many applications to drive highly compute-intensive (global) optimization and estimation processes.

The improvements themselves are due to a combination of several factors: The more accurate second-derivative rule allows us, in combination with a better error estimate, to take larger steps, which in turn allows us to reduce the number of otherwise expensive sensitivity calculations. The re-use of the Jacobians from the sensitivity calculations further reduces the total computational costs. Although each integration step is more expensive than in first-derivative methods, due to the

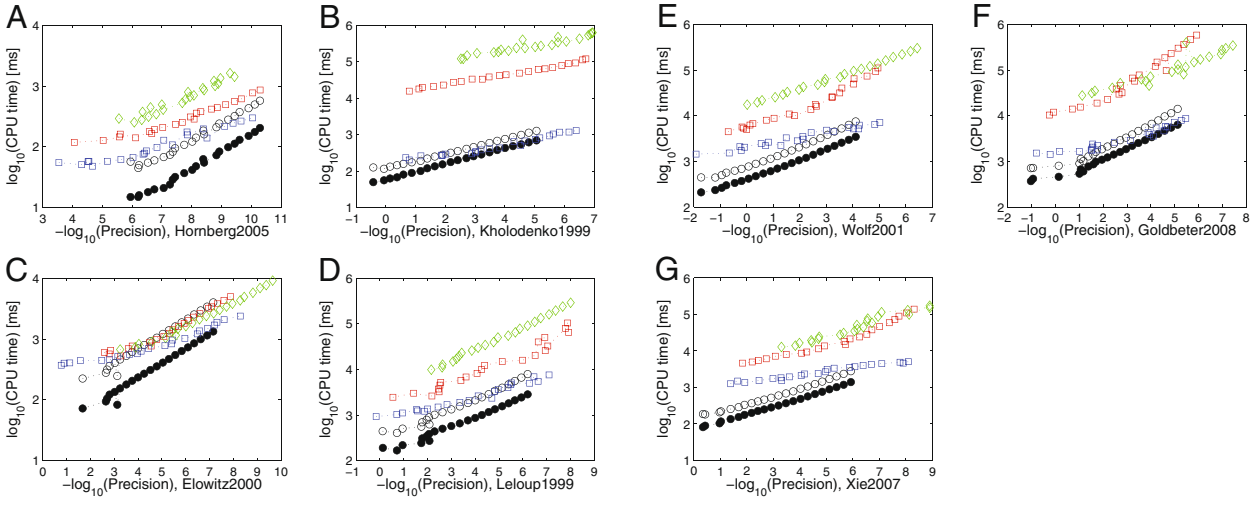

Figure 5 Precision/work diagrams with parameter sensitivities. Precision-work diagrams for integration with parameter sensitivities. (**A-G**) Computation times for all models for which the systems dynamics were solved with all ODE integrators (see X-axis for model specifications) as a function of precision using `odeSD` (open black circles), $\texttt{odeSD}_\texttt{mex}$ (filled black circles), `ode15s` (red squares), `radau5` (green diamonds), and `cvodes` (blue squares). The models were integrated for the time spans shown in Table 2.

additional second-derivative information that needs to be computed, far less steps are required in total, resulting in a more efficient method.

To be of practical relevance for applications in systems biology, `odeSD` and $\texttt{odeSD}_{\texttt{mex}}$ are accessible via Matlab interfaces[a], and we plan to make them more easily available through integrated modeling environments such as the SBToolbox2 [22] and COPASI [48], e.g. via the native C-language interface. To accelerate larger optimization and estimation processes, further efficiency improvements through the use of sparse matrix routines and by more elaborate step size control schemes are possible.

In terms of numerical algorithms, to our knowledge, this is the first practical application of a second-derivative integration method with good performance. Key, novel features of our integrator, such as a more precise error estimator and direct computation of parameter sensitivities with re-use of the Jacobians, may well be suited for other problems or types of integrators. Importantly, while our integrator has been developed for (bio)chemical and reaction networks, it is still quite general in targeting stiff and sparse ODE systems. Overall, we feel that a lot is to be gained by adapting general algorithms to specific problem domains, and that results from the work on specific problems can spill over to the broader field.

Availability

The integrator (Matlab and C-language versions) and the model conversion framework are available via `http://www.csb.ethz.ch/tools`.

Endnotes

[a]We note that it is not possible to execute $\texttt{odeSD}_{\texttt{mex}}$ in the 64-bit Windows version of Matlab R2010a as well as in 64-bit Linux versions prior to R2010a in our testing environments, for reasons of memory allocation problems in Matlab.

Additional files

Additional file 1: Supplementary Text and Figures for A Specialized ODE Integrator for the Efficient Computation of Parameter Sensitivities.

Additional file 2: Supplementary Source Code for A Specialized ODE Integrator for the Efficient Computation of Parameter Sensitivities.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

PG developed the second-derivative integration scheme for the parameter sensitivities and the new error estimate, as well as the final Matlab, `mex` and C-language versions. SD developed the automatic generation of the right-hand sides. LW implemented and tested the first versions of the integrator. JS designed and conducted the experiments. Both PG and JS wrote the manuscript, which was read and approved by all authors.

Acknowledgements

We thank Ernst Hairer for comments and discussions. This work was supported by the EU FP6 project BaSysBio (LSHG-CT-2006-037469), the EU FP7 project UniCellSys (HEALTH-F4-2008-201142), the Swiss Initiative for Systems Biology SystemsX.ch (RTD project YeastX), by an ETH Excellence Scholarship (LW), and by Swiss National Science Foundation's Individual Support Fellowships Nr. PBEZP2-127959 and Nr. PA00P2-134146 (PG).

Author details

[1]Mathematical Institute, University of Oxford, Oxford, UK. [2]Department of Biosystems Science and Engineering, ETH Zurich, 8092 Zürich, Switzerland. [3]School of Engineering and Computing Sciences, Durham University, UK.

Received: 27 April 2011 Accepted: 22 March 2012
Published: 20 May 2012

References

1. Chen WW, Niepel M, Sorger PK: **Classic and contemporary approaches to modeling biochemical reactions.** *Genes Dev* 2010, **24**(17):1861–1875. [http://dx.doi.org/10.1101/gad.1945410].
2. Gutenkunst RN, Waterfall JJ, Casey FP, Brown KS, Myers CR, Sethna JP: **Universally sloppy parameter sensitivities in systems biology models.** *PLoS Comput Biol* 2007, **3**(10):1871–1878. [http://dx.doi.org/10.1371/journal.pcbi.0030189].
3. Turányi T, Rabitz H: **Local methods.** In *Sensitivity analysis*. Edited by Saltelli A, Chan K, Scott E. Chichester, UK: John Wiley & Sons, Ltd.; 2000:81–100.
4. Banga JR, Balsa-Canto E: **Parameter estimation and optimal experimental design.** *Essays in Biochem: Syst Biol* 2008, **45:**195–209.
5. Cheng H, Sandu A: **Efficient uncertainty quantification with the polynomial chaos method for stiff systems.** *Math Comput Simul* 2009, **79**(11):3278–3295.
6. Doyle FJ, Stelling J: **Systems interface biology.** *J R Soc Interface* 2006, **3**(10):603–616.
7. Cao Y, Li S, Petzold L, Serban R: **Adjoint Sensitivity Analysis for Differential-Algebraic Equations: The Adjoint DAE System and Its Numerical Solution.** *SIAM J Sci Comput* 2003, **24**(3):1076–1089.
8. Wilkins AK, Tidor B, White J, Barton PI: **Sensitivity Analysis for Oscillating Dynamical Systems.** *SIAM J Sci Comput* 2009, **31**(4):2706–2732.
9. Vassiliadis VS, Canto EB, Banga JR: **Second-order sensitivities of general dynamic systems with application to optimal control problems.** *Chem Eng Sci* 1999, **54**(17):3851–3860.
10. Balsa-Canto E, Banga JR, Alonso AA, Vassiliadis VS: **Efficient Optimal Control of Bioprocesses Using Second-Order Information.** *Ind Eng Chem Res* 2000, **39**(11):4287–4295.
11. Sher A, Wang K, Wathen A, Mirams G, Abramson D, Gavaghan D: **A Local Sensitivity Analysis Method for Developing Biological Models with Identifiable Parameters: Application to L-type Calcium Channel Modelling.** In *IEEE Sixth International Conference on e-Science*; 2010:176–181.
12. Liu G, Swihart MT, Neelamegham S: **Sensitivity, principal component and flux analysis applied to signal transduction: the case of epidermal growth factor mediated signaling.** *Bioinformatics* 2005, **21**(7):1194–1202.
13. Chen WW, Schoeberl B, Jasper PJ, Niepel M, Nielsen UB, Lauffenburger DA, Sorger PK: **Input-output behavior of ErbB signaling pathways as revealed by a mass action model trained against dynamic data.** *Mol Syst Biol* 2009, **5**(239).
14. Barabási AL, Oltvai Z: **Network biology: understanding the cell's functional organization.** *Nat Rev Genet* 2004, **5:**101–13.
15. Hucka M, Finney A, Sauro H, Bolouri H, Doyle J, Kitano H, Arkin A, Bornstein B, Bray D, Cornish-Bowden A, Cuellar A, Dronov S, Gilles E, Ginkel M, Gor V, Goryanin I, Hedley W, Hodgman T, Hofmeyr J, Hunter P, Juty N, Kasberger J, Kremling A, Kummer U, Novere NL, Loew L, Lucio D, Mendes P, Minch E, Mjolsness E, Nakayama Y, Nelson M, Nielsen P, Sakurada T, Schaff J, Shapiro B, shimizu T, Spence H, Stelling J, Takahashi K, Tomita M, Wagner J, Wang J: **The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models.** *Bioinformatics* 2003, **19**(4):524–31.

16. Gupta GK, Sacks-Davis R, Tischer PE: **A Review of Recent Developments in Solving ODEs.** *ACM Comput Surv* 1985, **17:**5–47.
17. Cash JR: **Efficient numerical methods for the solution of stiff initial-value problems and differential algebraic equations.** *Proc R Soc London Series a-Math Phys Eng Sci* 2002, **459**(2032): 797–815.
18. Hindmarsh AC: **GEAR: Ordinary Differential Equation System Solver.** Technical Report UCID-30001, Rev. 3, Lawrence Livermore National Laboratory 1974.
19. Cohen SD, Hindmarsh AC: **CVODE, a stiff/nonstiff ODE solver in C.** *Comput Phys* 1996, **10**(2):138–143.
20. Hindmarsh AC, Brown PN, Grant KE, Lee SL, R S, Shumaker DE, S WC: **SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers.** *ACM Trans on Math Software* 2005, **31**(3):363–396.
21. Shampine LF, Reichelt MW: **The MATLAB ODE Suite.** *SIAM J Sci Comput* 1997, **18:**1–22.
22. Schmidt H, Jirstrand M: **Systems Biology Toolbox for MATLAB: a computational platform for research in systems biology.** *Bioinformatics* 2006, **22**(4):514–5. [Schmidt, Henning Jirstrand, Mats Research Support, Non-U.S. Gov't England Bioinformatics (Oxford, England) Bioinformatics. 2006 Feb 15;22(4):514-5. Epub 2005 Nov 29.].
23. Özyurt DB, Barton PI: **Cheap second order directional derivatives of stiff ODE embedded functionals.** *SIAM J Sci Comput* 2005, **26**(5):1725–1743.
24. Obrechkoff N: **Sur les quadrature mecaniques.** *Spisanic Bulgar Akad Nauk* 1942, **65:**191–289. [Reviewed in Mat. Rev. 10:70].
25. Enright WH: **Second derivative multistep methods for stiff ordinary differential equations.** *SIAM J Numer Anal* 1974, **11**(2):321–331.
26. Hairer E, Norsett S, Wanner G: *Solving Ordinary Differential Equations I.* Berlin: Springer Verlag; 1987.
27. Addison CA, Gladwell I: **Second derivative methods applied to implicit first- and second-order systems.** *Int J Numer Methods Eng* 1984, **20**(7):1211–1231.
28. Corliss G, Griewank A, Henneberger P, Kirlinger G, Potra F, Stetter H: **High-order stiff ODE solvers via automatic differentiation and rational prediction.** In *Numerical Analysis and Its Applications, Volume 1196 of* Lecture Notes in Computer Science. Edited by Vulkov L, Wasniewski J, Yalamov P. Berlin / Heidelberg; 1997:114–125.
29. Caracotsios M, Stewart WE: **Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations.** *Comput Chem Eng* 1985, **9**(4):359–365.
30. Feehery WF, Tolsma JE, Barton PI: **Efficient sensitivity analysis of large-scale differential-algebraic systems.** *Appl Numer Math* 1997, **25:**41–54.
31. Serban R, Hindmarsh AC: **CVODES, the sensitivity-enabled ODE solver in SUNDIALS.** In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol 6, Pts A-C*; 2005:257–269.
32. Machné R, Finney A, Müller S, Lu J, Widder S, Flamm C: **The SBML ODE Solver Library: a native API for symbolic and fast numerical analysis of reaction networks.** *Bioinformatics* 2006, **22**(11):1406–1407.
33. Hairer E, Wanner G: *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Berlin: Springer Verlag; 1991.
34. Engstler C: **Matlab implementation of the Radau IIA method of order5 by Ch. Engstler after the Fortran Code RADAU5 of Hairer/Wanner.** 1999. [`http://na.uni-tuebingen.de/projects.shtml`].
35. Serban R: **sundialsTB v2.4.0, a Matlab interface to Sundials.** Technical Report UCRL-SM-212121, Lawrence Livermore National Laboratory 2009.
36. Li C, Donizelli M, Rodriguez N, Dharuri H, Endler L, Chelliah V, Li L, He E, Henry A, Stefan MI, Snoep JL, Hucka M, Novère NL, Laibe C: **BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models.** *BMC Syst Biol* 2010, **4:**92 http://dx.doi.org/10.1186/1752-0509-4-92.
37. Hornberg JJ, Bruggeman FJ, Binder B, Geest CR, de Vaate AJMB, Lankelma J, Heinrich R, Westerhoff HV: **Principles behind the multifarious control of signal transduction. ERK phosphorylation and kinase/phosphatase control.** *FEBS J* 2005, **272:**244–258.
38. Kholodenko BN, Demin OV, Moehren G, Hoek JB: **Quantification of short term signaling by the epidermal growth factor receptor.** *J Biol Chem* 1999, **274**(42):30169–30181.
39. Singh A, Jayaraman A, Hahn J: **Modeling regulatory mechanisms in IL-6 signal transduction in hepatocytes.** *Biotechnol Bioeng* 2006, **95**(5):850–862.
40. Borisov N, Aksamitiene E, Kiyatkin A, Legewie S, Berkhout J, Maiwald T, Kaimachnikov NP, Timmer J, Hoek JB, Kholodenko BN: **Systems-level interactions between insulin-EGF networks amplify mitogenic signaling.** *Mol Syst Biol* 2009, **5:**256.
41. Ung CY, Li H, Ma XH, Jia J, Li BW, Low BC, Chen YZ: **Simulation of the regulation of EGFR endocytosis and EGFR-ERK signaling by endophilin-mediated RhoA-EGFR crosstalk.** *FEBS Lett* 2008, **582**(15):2283–2290.
42. Elowitz MB, Leibler S: **A synthetic oscillatory network of transcriptional regulators.** *Nature* 2000, **403**(6767):335–338.
43. Leloup Goldbeter: **Chaos and birhythmicity in a model for circadian oscillations of the PER and TIM proteins in Drosophila.** *J Theor Biol* 1999, **198**(3):445–459.
44. Wolf J, Sohn H, Heinrich R, Kuriyama H: **Mathematical analysis of a mechanism for autonomous metabolic oscillations in continuous culture of Saccharomyces cerevisiae.** *FEBS Lett* 2001, **499**(3):230–234.
45. Goldbeter A, Pourquié O: **Modeling the segmentation clock as a network of coupled oscillations in the Notch, Wnt and FGF signaling pathways.** *J Theor Biol* 2008, **252**(3):574–585.
46. Xie Z, Kulasiri D: **Modelling of circadian rhythms in Drosophila incorporating the interlocked PER/TIM and VRI/PDP1 feedback loops.** *J Theor Biol* 2007, **245**(2):290–304.
47. Maly T, Petzold LR: **Numerical methods and software for sensitivity analysis of differential-algebraic systems.** *Appl Numer Math* 1996, **20**(1-2):57–79.
48. Mendes P, Hoops S, Sahle S, Gauges R, Dada J, Kummer U: **Computational modeling of biochemical networks using COPASI.** *Methods Mol Biol* 2009, **500:**17–59. [http://dx.doi.org/10.1007/978-1-59745-525-1_2].

doi:10.1186/1752-0509-6-46
Cite this article as: Gonnet *et al.*: A specialized ODE integrator for the efficient computation of parameter sensitivities. *BMC Systems Biology* 2012 **6**:46.